Attorney's Docket No.: 16036-002001 / 02-11469

APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: EMAIL USING QUEUES IN NON-PERSISTENT MEMORY

APPLICANT: FRANK ADDANTE AND TIM MCQUILLEN

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EV 399312455 US

February 11, 2004
Date of Deposit

EMAIL USING QUEUES IN NON-PERSISTENT MEMORY

Cross Reference To Related Applications

[0001] This application claims the benefit of U.S. Provisional Application No. 60/449,301, filed on February 20, 2003, the contents of which are herewith incorporated by reference.

Background

[0002] E-mail is a well-established process of sending a text message from a sender to a recipient. The process of conventional e-mail is defined by a number of different interacting protocols and servers.

[0003] In conventional e-mail, a personal computer 100 runs an e-mail client 102. Well-known e-mail clients include Microsoft outlook and Outlook express. The e-mail client may be a standalone client, or may be a modified e-mail client running within a web page such as "Hotmail". The e-mail client lists the messages in the user's mailbox by headers, and allows the user to select and read the e-mail that is associated with that header. An e-mail client also allows creation of new messages and sending of the new messages.

[0004] The e-mail client communicates with an e-mail server 120 at the user's local Internet Service provider, here shown as "domain" 99, in order to send and receive messages over the Internet 110 or more generally over any network connection. A domain can include a single mail cluster, e.g., all of hotmail.com, or can include multiple mail clusters that are somehow associated.

[0005] The server receives e-mail messages from a client 102, and forms a list of those messages. The server 120 typically includes a processor or computer of some type, running the special email programs that are described herein.

[0006] The mechanics of the e-mail system operate by using three different protocols, known as SMTP, POP3 and IMAP. The SMTP server listens on port 25 to receive incoming emails. The POP3 server 140 handles delivery of local messages to the local mailboxes, such as 145. The mailbox is really a queue that is formed to provide the message an email client, when that client logs in to the POP3 server 140 at the domain 99.

[0007] Sending emails is handled differently than receiving emails. In order to send an e-mail, the e-mail client 102 interacts with the SMTP server 130 at the domain 99. If the message is intended for another mailbox within the same domain, then the SMTP server 130 sends the message to the local POP3 server 140. If the message is intended for another domain, the SMTP server communicates with a domain name server or DNS 135. The DNS stores a database, which is updated from the Internet, that stores the IP address for all domains. The DNS provides the IP address to the SMTP server 130.

[0008] The messages are stored on the hard drive 131 of the SMTP server 130. Software called a "picker" often operates to streamline the operations. The picker looks at messages stored on the SMTP server's hard drive 131, and carries out the mechanics of analyzing the message headers for destination, communicating with the DNS, and looking for an available port for the SMTP server on the destination domain.

[0009] Once these operations are completed, the SMTP server 130 sends the message using the IP address that it obtained from the DNS server 135 to another SMTP server 150 at the destination domain 149. The SMTP server 130 communicates with the corresponding SMTP server 155 at domain 149. The message is transferred to the SMTP server 155 at domain 149. Since SMTP server 155 recognizes that the message is for a local mailbox, it provides the message to its local POP3 server 160 which queues the message to a local mailbox 165.

[0010] Sending email often uses an open-source program known as sendmail™ which also includes many additional capabilities, including the ability to queue messages which cannot be sent immediately.

[0011] The receiving of emails uses POP3 server. The POP3 server maintains a collection of text files, one for each e-mail that has been sent or received. Each time a new e-mail is received, it adds that e-mail to its recipient file, or mailbox. The e-mail client 102 communicates with the POP3 server 140 at the local domain 99. The POP3 server provides the e-mail client 102 with contents of its mailbox, and then deletes the messages.

[0012] An IMAP server may be used in addition to or in place of POP3. IMAP allows the e-mail into folders which stay on the server.

[0013] For a large e-mail server, there may be many pickers operating at once, e.g. 50 to 100 pickers. Each of these pickers are obtaining information from the SMTP server's hard drive 131, moving messages one at a time from the hard drive.

Summary

[0014] It has been noticed that throughput limitations often occur in conventional e-mail systems of the type noted above. Certain limitations may be caused by the need to access files that are on the hard drive of the SMTP server.

[0015] The present system describes a messaging system that may operate in this conventional e-mail environment, but which may reduce delays caused by the operations. The inventor recognized that the operations on the hard drive of the SMTP server can burden the server and can also cause limits in throughput. According to an embodiment, the messaging system may store the mail in memory queues and memory queue maps that are formed within non-persistent, e.g., random access memory. A message transfer agent formats the messages and addresses, and organizes them to be sent from the memory. Therefore, for example, this system allows opening a pipe to a specific domain such as Hotmail, and then sending through as many messages as possible.

[0016] In an embodiment, e-mail messages are read directly from the database into memory and delivered to mailboxes or a host, without writing those e-mails back to persistent storage as part of the e-mail handling routine.

[0017] One aspect describes a message wrapper utility that divides the message into personalized and non-personalized parts.

[0018] Another aspect describes a crash prevention agent that stores the information within the non-persistent memory at intervals. The state of the application processing is thus saved to disk, in a very efficient way. This facilitates recovery in the event of a crash, while minimizing the amount of the information which is actually saved to disk.

Brief description of the drawings

[0019] These and other aspects will now be described in detail with reference to the accompanying drawings, wherein.

[0020] Figure 1 shows a diagram of e-mail flow;

[0021] Figure 2 shows a diagram of the queues that are formed according to the present system;

[0022] Figure 3 shows a flowchart of creating and handling the queues;

[0023] Figure 4 shows a flowchart of processing the messages within the queues;

[0024] Figure 5 shows a flowchart of one aspect of the load balancing;

[0025] Figure 6 shows a block diagram of the queues and agents handling the queues; and

[0026] Figure 7 shows the different functional elements that make up the operating program.

Detailed Description

[0027] The present system describes an improved e-mail handling and transferring system.

[0028] Initially, the description given herein is a description of software modules which run on a general-purpose computer, such as a workstation type computer or a computer based on the x86 architecture. However, it should be understood that the description is given herein could operate as hardware, for example based on the dedicated circuitry, or in FPGA components, with the functions being defined in hardware definition language. While the description given herein is of software, the inventors intend this description to similarly cover hardware devices which operate in comparable ways to that described herein.

[0029] Figure 2 illustrates the formation and use of a message queue map. The message queue map is preferably formed in non persistent storage, e.g., random access memory running within a mail server, e.g., the server performing the SMTP function. In a preferred embodiment, the entire mail processing operation occurs in non-persistent memory of this type.

[0030] Prior systems have taught away from using non-persistent memory for the email processing. In fact, the use of non-persistent memory could cause significant problems when and if the system crashes during operation. The present system, however, teaches a way to avoid loss of functionality during a crash, by storing information about which emails have been processed, and the state of processing of these emails. An aspect describes a very efficient way to save that state.

[0031] The queue map which is shown in figure 2 has a number of message queues shown respectively as 200, 202 and 204. Each of the message queues is defined based on various variables that may include domain name, systems user (in case of virtual servers). In simplest version of a queue, it is associated with a specific domain. For example, message queue 200 is associated with domain 210 which is Hotmail.com. Message queue 204, intended for yahoo.com (domain 214), includes two different data nodes 252, 254, which are each intended for delivery to domain yahoo.com. Each data node represents personalized information about the e-mail to be sent.

[0032] A new message 220 is also shown. As part of the operation, this new message needs to be added into the existing queue map. An input handler shown as 225 may operate as shown in the flowchart of Figure 3. At 300, the process receives the new message 220. The input handler operates, at 305, to create a data node 230 as a digest representing the message. A data node in this embodiment is an object that represents one message. The data node includes information about the message being sent; and may include the recipient, sender, data about the email, domain, a unique session identifier, visit count, and other information for Quality of Service ("QoS") guarantees and routing specifics. Note that the nodes are not the emails themselves - rather they are just pointers to the emails as stored in memory.

[0033] The data node is analyzed to determine the appropriate queue (function of domain and miscellaneous variables) at 310. 315 determines if message can be appended to existing . If so then the data node is appended to the existing queue at 320.

[0034] For example, here the new message 220 is intended for the domain 210 of Hotmail.com. Therefore, the data node 230 is appended to the end of the existing queue 200.

[0035] In the alternative, if the node cannot be put into an existing queue at 315, then a new queue is created at 325, and the data node 230 is appended to the newly created queue at 330. In this way, multiple queues are formed, each relating to nodes representing messages with similar routing strategies. Multiple queues may be provided for each queue variable if the existing queue has more than a maximum number of messages.

[0036] Since each queue represents messages that will require the same processing strategy such as delivery to the same domain, the entire queue of messages can be sent at once, thereby streamlining the sending process.

[0037] A command set, shown here as the output handler 240, can operate on the messages and queues according to the flowchart of Figure 4. Each data node represents a particular message. The output handler processes the data node in order to send the mail to the intended recipient. The output handler 240 is shown as being a single process, but multiple processes may be operated at once, with, for example, each process handling a single queue at any one time or using a pipelined or multi-threaded system.

[0038] The process starts at 400, where the output handler looks for the next queue to process. This may be done in round-robin fashion, where each queue is assigned a number (n) for example, and the system simply looks for n+1 queue, where a maximum n of queues can exist. An alternative system is that the output handler always handles the queue with the greatest number of message nodes therein. In this example, queues are sorted according to their length and in that case, 400 finds the longest queue or the next full queue. In another embodiment, however, the amount of time that the queue has existed may also be taken into account. Another words, the longest queue would be sorted first unless that longest queue had not been processed for a specified time such as X minutes.

[0039] In a current queue at 420, the message is found, removed from the queue and processed for delivery at 420. A message wrapper is formed at 425, which may include multiple messages within the wrapper. Each of the messages within the wrapper has its own personalized content, but the common parts of the messages (such as the domain information) are provided by the wrapper itself. This may provide further streamlining of the process.

[0040] 430 determines if there are more items with the queue that can go within the same wrapper, and if so, gets the message at 420 and adds it to the wrapper at 425. After the queue is finished, processing is carried out by 435 which shows locating an SMTP server for the recipient domain, making a connection to the SMTP server, sending protocol tokens representing the message, and then delivering the messages.

[0041] Once all of the messages in the queue have been removed, then the queue is removed from the memory map, or extinguished, at 445. If message processing fails due to a recoverable error, then the message may be pushed back into the queue, or into a new queue indicative of the same domain. Each time the message delivery fails, the "visit count" is incremented. Message failure may occur due to the recipient servers being unavailable, e.g. busy or inaccessible. Processing then moves to the next queue at 450.

[0042] An overflow prevention is shown in the flowchart of figure 5. Overflow may occur if messages are received faster than the queuing agent 225 can handle the messages. At 500, the input handler detects that it has a backlog which is greater than a specified amount, for example, 1000 messages, or if the size of the message queue will take longer than a specified time to process, such as 3 seconds. At 505 the input handler sends a "Pause" acknowledgment to the message server that is receiving the messages, e.g., the SMTP server. The pause acknowledgment indicates to the message server that it should stop sending messages that it has received. This causes the message server to save the received messages until the backlog is reduced.

[0043] The loop continues to check the backlog at 500. If the number of messages in the queue has fallen below the size limit at 500, then the relay server sends a "send" acknowledgement to the message server at 510. This allows the message server to start sending messages again.

[0044] The server application 702 may be able to transmit certain e-mail messages to certain domains quicker than others. In this case, the load balancer 732 may pass more e-mails for the faster domains to the server 702 than it does for the slower domains. The statistics polling process 764 and listener processes may carry this out. In this way the message server effectively adjusts the feed rate according to the rate at which the relay servers are performing. This may allow the system to take into account slower relay servers, and prevent blocking of messages by slower relay servers.

[0045] The basic load-balancing is made by the following steps:

get the next message parameter;

generate the full header and body with personalization;

query the current load conditions and compute delivery rates for the messages;

compute whether the system relay is ready for new messages; and

if so, push the messages.

[0046] In this way, the input handler receives the messages only when it can handle the messages. This provides one aspect of basic load balancing.

[0047] Figure 6 shows a basic block diagram of the e-mail messaging system. A relay server 602 carries out the functions of handling the input messages. The relay server 602 includes a queuing agent 604, as well as a memory queue statistics element 606 and a message reactor 607. The message reactor 607 can be an SMTP server or a server that carries out comparable functions or a device that interfaces to an existing SMTP server.

[0048] The message reactor 607 delivers the messages to the user mailboxes 608. Bounce server 606 detects any messages that are intended for mailboxes which do not exist, and "stores" those messages for further processing.

[0049] An input parser 616 receives and characterizes the messages. Parser 616 includes a message server 610, statistics collector 612 and recovery agent 614. Each of these functions can be carried out in RAM or nonpersistent storage in order to facilitate the processing. The recovery agent stores snapshots of system variables to persistent storage, to allow the system to recover from a computer crash. In an alternative, although perhaps less preferred embodiment, however, queues or parts of queues can be maintained on a disk drive. The functions of these elements will be described in further detail herein.

[0050] Figure 7 is a flowchart of the passing of messages between the client, server and license server authentication unit.

[0051] The client application is shown as 700, and this may be for example, an e-mail client operating in a user's computer. The client application 700 creates personalized messages of a conventional type, and controls and commands sending those messages to the server. This is done using a number of agents; all of which may operate in software.

[0052] In the client application 700, a number of different processes cooperate together in order to form and send an e-mail. A message files database 712 represents the specific message files which are being sent, that is, the text and/or attachments that form the unique parts of the e-mail. The other parts that may be used for many different e-mails, are stored in a file, such as databases 710,714. A database 710 provides the e-mail addresses. Mailing preferences, including mailing information and the like for the recipient of the e-mail, are stored in the database 712. These two databases are used in conjunction with the mailing configuration file 714 that stores tracking information for the resulting e-mail.

[0053] The nodes, such as 252 shown in figure 2, may be pointers to areas in the message file 712 and/or areas in the database 710 and configuration file 714. In addition, each e-mail may be assigned with a unique ID formed from a bit vector of the queued e-mail. The bit vector may be stored in persistent storage. The bit vector may include sufficient information to reconstruct the message and the state of processing of each of the e-mails. This information is stored on disk or persistent storage. This enables recovering the entire state of processing of the system, without storing the entirety of the e-mails to disk.

[0054] Only certain data representing the contents of the e-mails, the locations in memory, and the like are stored. For example, the message IDs of each of the messages in each of the queues may be recorded periodically in order to save the state. If the relay server goes down for any reason, then the IDs of the messages that were currently being processed are recorded. The crash is remedied by starting a new message server to transfer these non-processed messages to other relay servers in the set up.

If all relay servers go down simultaneously, the undelivered message IDs remain recorded, and can be sent by the system.

[0055] The e-mail address records in the database 710 are processed by a database parser 720 along with the contents of the mailing configuration file 714. Correspondingly, the message records in the message database 712 are processed by a message parser 722. The two parsers 720 and 722 act on the database records and pass parsed content to the personalization agent 724. This agent combines the parsed information from the address record with the parsed message. In one embodiment, the parsed message may be formed by a message template, filled in with tokens from the message list parameters, for example the parameters that are shown in element 230 in figure 2. The personalized content such as the name or other information is inserted into the output messages based on the personalization.

[0056] An e-mail message is created using the contents of the personalization agent 724 to create a message wrapper, which includes the message from the message parser 722 within the parsed address from the database parser 720 to create a personalized message 725. The resulting address is preferably manipulated solely within random access memory to enable quicker handling.

[0057] The personalized message 725 is then passed to a queuing agent 730 which takes the message input and queues it in the appropriate queues. The queues may be formed as described previously with reference to figure 2. The e-mail is generally queued according to the domain that will receive the e-mail. Often used domain names may receive multiple queues.

[0058] A client-to-server Load balancer 732 monitors the queues to ensure that the server 702 is not overwhelmed by incoming e-mails, as described above with reference to figure 5.

[0059] The server application 702 uses request handler 740 to take the messages and deliver the messages to one or more delivery agents 742. While only one delivery agent is shown, there may be many such agents. These delivery agents 742 communicate the e-mail messages to a remote connection pool manager 744 that manages a number of remote connections. The remote connection pool manager 744 establishes, maintains and terminates connections with remove SMTP servers shown as 746, 780 and 782.

[0060] The remote connection pool manager 744 may maintain the connections with the recipient SMTP servers directly;

taking the burden of doing this off of the SMTP server at the local ISP.

[0061] The remote connection manager also uses asynchronous DNS resolvers 750 which operate from an off-line queue or cache 352 that is periodically updated. The DNS lookup may be asynchronous relative to the remaining parts of the message delivery. In this way, the lookup of DNS information at 750 from the DNS cache 752 can be performed in parallel with other parts of the message delivery.

[0062] The delivery agent or agents 342 are also in communication with the logging agent 760, which forms a monitor process to monitor which e-mails that have been sent. This may enable complete recovery in the case of a system crash.

[0063] The mailing configuration file 714, the database parser 720 and message parser 722 operate on a predictable and logical basis (Flowchart?). Therefore, by knowing where e-mail transmission was interrupted, the point that existed at the time of any system crash can easily be recovered.

[0064] The logging agent 760 also communicates with the statistics listener processes 762 and the statistics polling process 764. The logging agent 760 monitors successful and unsuccessful e-mail transmissions.

Unsuccessful transmissions may occur when a remote server is unavailable or an unsuccessful DNS resolution occurs. The status polling process 764 is also in communication with the queuing agent 730 and maintains a record of the last outgoing message. In this way, an interrupted mail stream may be re-established at the point of interruption.

[0065] The listener process 762 logs or provides information about successful e-mail transmissions. In an embodiment, the information about e-mails, in the list, is listed by reference only. For example, the listener process 762 may indicate which can successfully delivered. This also maintains information or logs about rejected e-mails. The e-mails may be rejected in complete form, so that forensic analysis may be performed to determine how the failure arose.

[0066] The logging agent 760 may also indicate when the e-mail was clicked on or opened.

[0067] The logging agent collects and aggregates e-mail information from multiple sources may also be in contact with a license server statistics collector 770, and also in contact with a license server authentication process 704. The license server authentication is also in communication with both the server and client.

[0068] The concept of license authentication is entirely a new paradigm according to the present system. The statistics collector 370 collect statistics about the number of messages that are processed by this system. In an embodiment, the server authentication 304 determines whether a user has paid appropriate license fees sufficient to cover the number/type of messages which have been sent. The server authentication 304 may refuse to send messages or may send warnings based on the number of messages having been exceeded. In this way, this software may operate effectively as pay-per-use software. That is, the initial software may be sold with some number of messages enabled. This may enable users to evaluate the software, almost as shareware, for a certain period of time. They may install it, and it will operate as desired until the specified number of messages is reached. After that, the user needs to pay additional license fees to process additional messages.

[0069] Although only a few embodiments have been described in detail above, other modifications are possible. Many of the modifications are described herein.

[0070] In an embodiment, the program features described above are written in C++ although it should be understood that other programs could be used to write this program, and also that the operations could be carried out using hardwired logic defined using hardware definition language, or FPGAs or other components.

[0071] While this system is described in the context of use with e-mail applications, it can be used or any system that sends messages, preferably text messages, over a network such as the Internet.

[0072] This system as embodied does not rely on third-party mail sending applications such as sendmail or other programs, although such can be used.